Лабораторная работа №7. Сервлеты

Теоретическая часть

Сервлет является Java-программой, выполняющейся на стороне сервера и расширяющей функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. Сервлет - это диспетчер процесса. Он находится на Web-сервере и обрабатывает входящие запросы и исходящие ответы.

Сервлеты должны реализовывать Servlet интерфейс, который определяет его методы жизненного цикла. Хотя сервлеты могут обслуживать любые запросы, они обычно используются для расширения веб-серверов. Для таких приложений технология Java Servlet определяет HTTP-специфичные сервлет классы.

Пакеты javax.servlet и javax.servlet.http обеспечивают интерфейсы и классы для создания сервлетов.

Жизненный цикл сервлета

Жизненный цикл сервлета состоит из следующих шагов:

1. В случае отсутствия сервлета в контейнере.

а) Класс сервлета загружается контейнером.

б) Контейнер создает экземпляр класса сервлета.

в) Контейнер вызывает функцию init(). Эта функция инициализирует сервлет и вызывается в первую очередь, до того, как сервлет сможет обслуживать запросы. За весь жизненный цикл функция init() вызывается только однажды.

2. Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает функцию service() для каждого запроса. Эта функция определяет тип пришедшего запроса и распределяет его в соответствующую для этого типа функцию для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих функций. Если поступил запрос, функция для которого не реализована, вызывается функция родительского класса и обычно завершается возвращением ошибки инициатору запроса.

3. В случае если контейнеру необходимо удалить сервлет, он вызывает функцию destroy(), которая снимает сервлет из эксплуатации. Подобно функции init(), эта функция тоже вызывается единожды за весь цикл сервлета.

В качестве контейнера может выступать, например, Tomcat или GlassFish.

Tomcat

Tomcat (в старых версиях — Catalina) — программа-контейнер сервлетов, написанная на языке Java и реализующая спецификацию сервлетов и спецификацию JavaServer Pages (JSP), которые являются стандартами для разработки веб-приложений на языке Java.

Tomcat позволяет запускать веб-приложения, содержит ряд программ для самоконфигурирования. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в сервере приложений Jboss.

Сервлеты не являются единственным методом работы с Web-страницами. Одной из самых ранних технологий для этой цели являлся CGI (common gateway interface), но он инициализировал отдельный процесс для каждого запроса, что не было очень эффективно. Существовали также патентованные серверные расширения, например Netscape Server API (NSAPI), но они были, да-да, патентованными. В мире Microsoft существует стандарт ASP

(active server pages). Сервлеты являются альтернативой всем этим технологиям и предлагают несколько преимуществ:

- Они также платформо-независимы, как язык Java.
- Они предоставляют полный доступ ко всем API языка Java, включая библиотеки для доступа к данным (например, JDBC).
- Они (в большинстве случаев) в своей основе более эффективны, чем CGI, поскольку сервлеты порождают новые субпроцессы (thread) для запросов, а не отдельные процессы.
- Существует распространенная отраслевая поддержка сервлетов, включая контейнеры для наиболее популярных Web-серверов и серверов приложений.

Сервлеты являются мощным дополнением к инструментальным средствам профессионального программиста.

Разработка сервлетов

При создании Java-сервлета обычно создается подкласс HttpServlet. Этот класс имеет методы, предоставляющие доступ к конвертам запроса и ответа для обработки запросов и создания ответов.

HTTP-протокол, естественно, не связан с Java. Это просто спецификация, определяющая, какими образом осуществляются запросы и ответы. Классы Java-сервлетов заключают эти низкоуровневые конструкции в Java-классы, имеющие удобные функции, которые облегчают работу с этими конструкциями в контексте языка Java. Когда пользователь инициирует запрос через URL, классы Java-сервлетов преобразуют их в HttpServletRequest и передают адресату, указанному в URL, как определено в конфигурационных файлах конкретного контейнера сервлетов, который вы используете. Когда сервер завершает свою работу с запросом, Java Runtime Environment упаковывает результаты в HttpServletResponse и затем передает HTTP-ответ обратно клиенту, пославшему запрос. Взаимодействуя с Web-приложением, вы обычно посылаете несколько запросов и получаете несколько ответов. Все они работают в контексте сессии, которая в языке Java заключается в объект HttpSession. Вы можете обратиться к этому объекту при обработке запросов и добавить что-нибудь к нему при создании ответов. Он обеспечивает некоторого рода контекст кроссзапросов.

Контейнер, например Tomcat, управляет средой исполнения для сервлетов. Вы можете настроить способ функционирования J2EE-сервера и должны настроить его для отображения ваших сервлетов внешнему миру. Как мы увидим в дальнейшем, используя различные конфигурационные файлы контейнера, вы обеспечиваете мост от URL (введенного пользователем в браузере) к серверным компонентам, обрабатывающим запрос, в который транслируется URL. Во время работы вашего приложения контейнер загружает и инициализирует ваш сервлет (сервлеты) и управляет его жизненным циклом.

Когда мы говорим о том, что сервлеты имеют жизненный цикл, это просто означает, что при активизации сервлета все работает предсказуемо. Другими словами, для любого создаваемого вами сервлета всегда будут вызываться определенные функции в определенном порядке. Вот обычный сценарий:

1) Пользователь вводит URL в браузере. Конфигурационный файл вашего Web-сервера указывает, что этот URL предназначен для сервлета, управляемого контейнером сервлетов на вашем сервере.

2) Если экземпляр сервлета еще не был создан (существует только один экземпляр сервлета для приложения), контейнер загружает класс и создает экземпляр объекта.

3) Контейнер вызывает функцию init() сервлета.

4) Контейнер вызывает функцию service() сервлета и передает HttpServletRequest и HttpServletResponse.

5) Сервлет обычно обращается к элементам запроса, передает запрос другим серверным классам для выполнения запрошенной службы и для доступа к таким ресурсам, как базы данных, а затем создает ответ, используя эту информацию.

6) При необходимости, когда сервлет выполнил полезную работу, контейнер вызывает функцию destroy() сервлета для его финализации.

"Выполнение" сервлета аналогично выполнению Java-программы. Если вы настроили ваш контейнер, и он знает о вашем сервлете, а также знает, что для определенных URL контейнер должен активизировать сервлет, контейнер будет вызывать функции жизненного цикла сервлета в предопределенном порядке.

Поэтому выполнение сервлета, в сущности, означает его корректную настройку и указание в браузере нужного URL. Естественно, логика реализована в коде сервлета. Вы не должны беспокоиться о низкоуровневых операциях, которые выполняются, пока что-то не перестает работать правильно.

К сожалению, что-то перестает работать правильно удручающе часто, особенно при настройке сервлета. Самым большим источником головной боли при создании приложений с сервлетами являются конфигурационные файлы. Их нельзя эффективно отлаживать. Настраивать их на правильную работу вы должны методом проб и ошибок, путем расшифровки сообщений об ошибках, которые можете увидеть (а может и нет) в вашем браузере.

```
public class HelloWorldServlet extends HttpServlet {
 public void service(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
  PrintWriter writer = response.getWriter();
  writer.println("Hello, World!");
  writer.close();
 }
}
```

Обратите внимание на то, что мы создали подкласс HttpServlet и переопределили функцию service(). Функция service() является самой главной функцией обработки, которую контейнер сервлетов будет вызывать в цикле жизни нашего сервлета. Она принимает конверт запроса и конверт ответа, к которым мы можем обратиться в нашей функции. Однако в данном случае нам не нужно этого делать, поскольку мы делаем только самое основное для того, чтобы сервлет заработал. Мы могли бы переопределить функцию doGet(), но service() дает все, что нам надо.

В нашей функции service() мы вызвали getWriter() нашего конверта ответа, для того чтобы разрешить вывод строкового литерала в поток. Затем мы закрыли поток. Это типично для сервлетов, выполняющих вывод информации: вы выполняете необходимую логику, а затем пишете результаты в выходной поток.

"Выполнение сервлета" заключается в запуске Tomcat и указании в Web-браузере URL для его активизации.

На заре Web-разработки многие профессиональные программисты должны были определять, как эффективно использовать сервлеты. Наиболее часто это приводило к бурному росту числа сервлетов на сервере. На каждый тип запроса приходился один сервлет.

Это быстро стало неудобным, поэтому программисты начали включать в свои сервлеты условную логику, которая делала их более адаптируемыми, обрабатывающими несколько типов запросов. Со временем это тоже стало приводить к плохому коду. Существует лучший способ, называемый сервлетом действия (action servlet), который реализует концепцию под названием Model 2.

Сервлет действия не имеет условной логики, выбирающей поведение сервлета. Вместо этого у вас имеются действия (определенные программистом классы), которым сервлет передает полномочия для обработки запросов различного типа. В большинстве случаев это намного лучший объектно-ориентированный подход, чем использование нескольких сервлетов или нескольких условий if в одном сервлете.

Правила использования сервлетов

1) Используйте один сервлет. Если вы не можете использовать только один сервлет, используйте минимальное их количество. Фактически, желательно использовать один, пока это не будет абсолютно невозможно. Вам не нужна сотня сервлетов. И, определенно, вам не нужен один сервлет для каждого типа запроса.

2) Не задерживайте работу в сервлете. Тратьте так мало времени в вашем сервлете, насколько это возможно. Сервлеты - это не место для бизнес-логики. Это плохой пример объектно-ориентированного дизайна. Представляйте сервлет как одну из следующих сущностей:

- Дополнительный уровень за вашим UI, помогающий получать "события" для сервера.

- Дополнительный уровень перед сервером, позволяющий использовать браузер в качестве UI.

В любом случае, это место, где вы быстро выполняете диспетчеризацию для других частей вашего приложения и покидаете сервлет.

Используйте действия. Система действий (action framework), даже совсем простейшая, является мощным методом. Она позволяет вам следовать предыдущему совету: тратить минимально возможное количество времени в вашем сервлете. Это также хорошо соответствует объектно-ориентированному дизайну. Каждый класс action делает либо одно действие, либо очень цельный набор связанных между собой действий.

3) Используйте service() до тех пор, пока больше не сможете это делать.

4) Не смешивайте бизнес-логику и логику представления. Генерирование сложных HTML-строк для вывода в поток для JSP-страницы хорошо для простого приложения, но при создании более функционального приложения этот способ не годится. Разумнее хранить логику представления в том месте, где она должна быть: в странице. JSP-технология позволяет вам делать это, но, как уже говорилось, требуется значительно больше работы для отделения бизнес-логики от логики представления.

5) Обрабатывайте исключительные ситуации должным образом. Нет ничего более разочаровывающего для пользователя, чем вид в браузере зашифрованной трассировки стека при возникновении чего-то непредвиденного на сервере.

Пример. Простой сервлета.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns = {"/ServletAppl"})
public class Servlet1 extends HttpServlet {

    static String ast;
    static boolean b;
    static long counter;
    static int cycle;

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
     * methods.
```

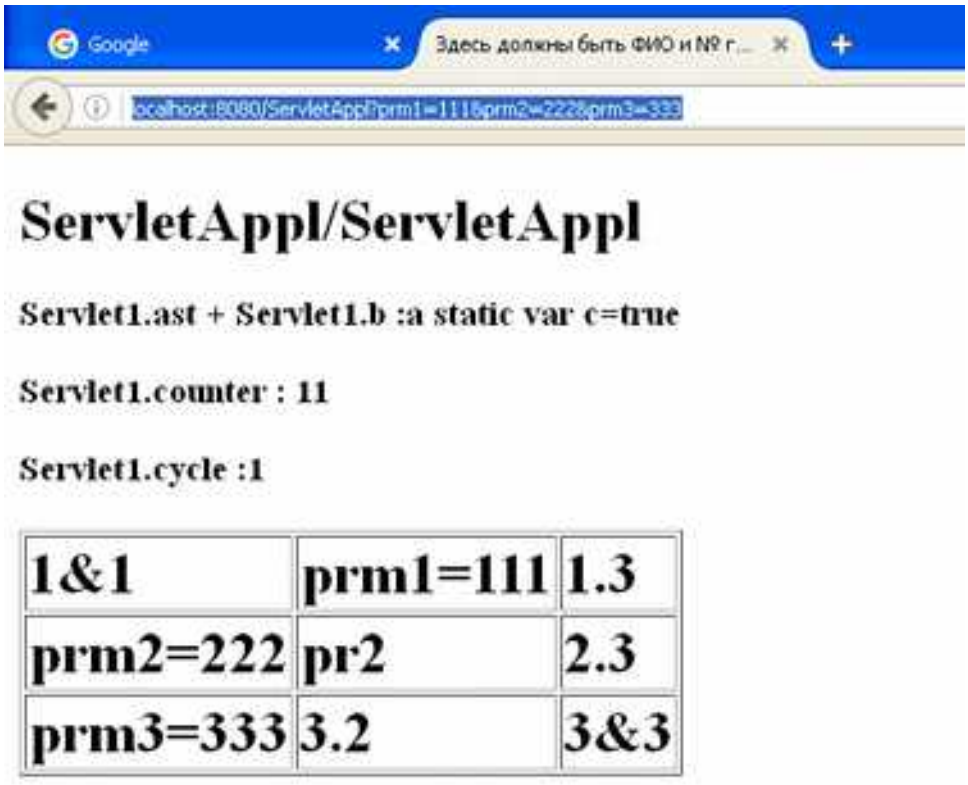
```
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */

public Servlet1(){
    Servlet1.ast = "a static var c=";
    Servlet1.b = false;
    Servlet1.counter = 0;
    Servlet1.cycle = 0;
}

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Здесь должны быть ФИО и № группы</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>ServletAppl" + request.getServletPath() + "</h1>");
        //==============I

        if (Servlet1.b) Servlet1.b = false;
        else Servlet1.b = true;
        Servlet1.counter++;
        if(Servlet1.cycle < 5){
            Servlet1.cycle ++;
        }else{
            Servlet1.cycle = 1;
        }

        out.println("<h3> Servlet1.ast + Servlet1.b :" + Servlet1.ast + Servlet1.b + "</h3>");
        out.println("<h3> Servlet1.counter : " + Servlet1.counter + "</h3>");
        out.println("<h3> Servlet1.cycle :" + Servlet1.cycle + "</h3>");

        String[][] tbl = new String[3][3];
        for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            tbl[i][j] = Integer.toString(i+1) + "&" + Integer.toString(j+1);
            }
        }

        String prm1 = request.getParameter("prm1");
        String prm2 = request.getParameter("prm2");
        String prm3 = request.getParameter("prm3");


        out.println("<h" + Servlet1.cycle + "><table border>"+
                "<tr>" + "<td>" + tbl[0][0] + "</td>" + "<td>prm1=" + prm1 + "</td>" + "<td>1.3</td>" +
                "<tr>" + "<td>prm2=" + prm2 + "</td>" + "<td>pr2</td>" + "<td>2.3</td>" +
                "<tr>" + "<td>prm3=" + prm3 + "</td>" + "<td>3.2</td>" + "<td>" + tbl[2][2] + "</td>" +
                "</tr>"
                + "</table></h" + Servlet1.cycle + ">");

        //==============II
        out.println("</body>");
        out.println("</html>");
    }
}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
```

```
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>

}
```

Некоторые настройки проекта

В свойствах проекта, в категории «Выполнение» очистить поле «Контекстный путь».

Примерный вид браузера



Практическая часть

1. В зависимости от варианта выполняется разработка web-приложения, т.е. сервлета. Примеры приведены выше.
2. Для создания web-приложения в NetBeans IDE нужно выбрать в меню пункт «Файл\Создать проект», в появившемся диалоговом окне выбрать категорию «Java Web» и соответствующий этой категории тип проекта «Веб-приложение». Нажать «Далее» и в появившейся вкладке «Имя и расположение» ввести название проекта. Снова нажать «Далее» в появившейся вкладке «Сервер и параметры настройки» указать сервер «Tomcat» (либо GlassFish) и версию «Java EE 7 web». Также на этой вкладке может быть изменён «контекстный путь». Нажать «Готово».
3. Следующий шаг - создание сервлета в web-приложении. Для этого нужно вызвать контекстное меню при нажатии правой клавиши мыши на имени проекта вашего web-приложения (которое обычно располагается слева, в окне «Проекты"). В этом меню нужно выбрать «Новый\Сервлет…». На вкладке «Имя и расположение» задать имя класса сервлета. Нажать «Готово».
4. Необходимо обеспечить передачу параметров в сервлет через строку url-запроса.
5. При перезагрузке страницы сервлета должно отображаться: 0 – изменение триггера, хранящегося в объекте сервлета; 1 – значение счётчика обращений к странице сервлета после его запуска.
6. Организовать вывод результатов работы сервлета: 0 - строки следуют одна за другой сверху вниз (таблица без видимых границ, состоящая из одного столбца и множества строк); 1 – данные полученные от сервлета должны быть каким-то образом размещены в видимой таблице, в таблице допускается произвольное число столбцов и строк.
7. Реализовать при обновлении страницы сервлета: 0 - уменьшение размера текста в таблице до заданной величины, после чего на странице должна появляться надпись (не в таблице), информирующая о том, что дальнейшее уменьшение не возможно; 1 - увеличение размера текста в таблице до заданной величины, после чего на странице должна появляться надпись (не в таблице), информирующая о том, что дальнейшее увеличение не возможно.
8. Реализовать возможность сброса размера текста в таблице через параметр строки url-запроса: 0 - до значения по умолчанию, 1 – до указанного значения.
9. Среди параметров, передаваемых в сервлет, нужно передавать Ф.И.О студента, выполнившего разработку сервлета, и номер его группы, которые должны отображаться следующим образом: 0 - в заголовке web-страницы возвращаемой сервлетом клиенту; 1 - на web-странице (не в таблице).
10. Взять за основную функцию, которую вычисляет сервлет, реализованную в первой лабораторной работе.
11. При необходимости могут быть изменены порты, по которым контейнер сервлетов Tomcat слушает запросы. Для изменения портов нужно в среде NetBeans войти в меню «Сервис\Серверы»: 0 — оставить порт по умолчанию, 1 — изменить на произвольный.

Варианты заданий

Таблица №1

№	(П.5)	(П.6)	(П.7)	(П.8)	(П.9)	(П.11)
1.	1	0	1	1	1	1

Число в указанном варианте следует толковать как двоичное число, каждый разряд которого соответствует столбцу из таблицы 1.

1	**47(101111)**	17	49	33	56	49	21
2	1	18	28	34	53	50	8
3	54	19	19	35	46	51	3
4	32	20	38	36	62	52	8
5	2	21	34	37	18	53	64
6	27	22	37	38	29	54	28
7	64	23	32	39	26	55	10
8	51	24	34	40	4	56	15
9	16	25	13	41	15	57	5
10	38	26	57	42	8	58	35
11	35	27	7	43	56	59	16
12	7	28	21	44	14	60	49
13	12	29	40	45	33	61	21
14	57	30	7	46	50	62	9
15	52	31	61	47	37	63	60
16	24	32	60	48	32	64	51